Automated Optimal OSP Mechanisms for Set Systems

The Case of Small Domains

Diodato Ferraioli[1][0000−0002−7962−5200]⋆, Adrian Meier[2], Paolo Penna[2], and Carmine Ventre[3][0000−0003−1464−1215]

[1] Università degli Studi di Salerno, Italy
dferraioli@unisa.it
[2] ETH Zurich, Switzerland
meiera@student.ethz.ch, paolo.penna@inf.ethz.ch
[3] King's College London, UK
carmine.ventre@kcl.ac.uk

Abstract. Obviously strategyproof (OSP) mechanisms have recently come to the fore as a tool to deal with imperfect rationality. They, in fact, incentivize people with no contingent reasoning skills to "follow the protocol" and be honest. However, their exact power is still to be determined. For example, even for settings relatively well understood, such as binary allocation problems, it is not clear when optimal solutions can be computed with OSP mechanisms.
We here consider this question for the large class of set system problems, where selfish agents with imperfect rationality own elements whose cost can take one among few values. In our main result, we give a characterization of the instances for which the optimum is possible. The mechanism we provide uses a combination of ascending and descending auctions, thus extending to a large class of settings a design paradigm for OSP mechanisms recently introduced in [9]. Finally, we dig deeper in the characterizing property and observe that the set of conditions can be quickly verified algorithmically. The combination of our mechanism and algorithmic characterization gives rise to the first example of automated mechanism design for OSP.

Keywords: Extensive Form Mechanisms · Bounded Rationality.

1 Introduction

The role of incentives in the design of algorithms has been a very active research area in the last two decades. Mechanism design has as its main objective the alignment of the objectives of the designer (e.g., optimality of the solution) with those of self-interested agents (e.g., maximize their utility). The crucial

⋆ This author is partially supported by GNCS-INdAM and by the Italian MIUR PRIN 2017 Project ALGADIMAR "Algorithms, Games, and Digital Markets".

assumption made in the area is that these self-interested agents have perfect rationality: they will be able to ascertain that there is no point in strategizing, whenever the mechanism is proved to be truthful (a.k.a., strategyproof (SP)).

Unfortunately this might be too strong an assumption for practical applications of these theoretically sound mechanisms. Even for the well-known second-price auction, bidders lie when submitting sealed bids but are truthful when the mechanism is implemented via an ascending auction [14]. Intuitively, this means that it is easier to understand how to play the latter implementation of Vickrey auction, whilst the former can be confusing for agents with imperfect rationality. The recent definition of *obviously strategyproof (OSP)* mechanisms [16] formalizes how a different (extensive-form) *implementation* can make it *obvious* for an agent to decide what strategy to adopt. Roughly speaking, in OSP mechanisms, the utility for the *worst scenario* when truth-telling is at least as good as that of the *best scenario* when cheating. Li [16] proves that OSP mechanisms are obvious to understand for people without any contingent reasoning skill.

Research about the power of OSP mechanisms has barely scratched the surface. While it is clear that ascending/descending price auctions are OSP (as it is obvious for a bidder to decide whether to quit the auction or not), few general paradigms are known for the design of OSP mechanisms that return "good" (e.g., optimal) solutions. *Deferred-acceptance* (DA) mechanisms [18] are OSP (as they essentially are ascending price auctions), but unfortunately their performance (approximation guarantee) for several optimization problems is quite poor compared to what strategyproof mechanisms can do [6, 9]. A combination of ascending and descending auctions has recently been given in [9] for the well-known scheduling related machines problem; the mechanism and its analysis rely on a generalization of the cycle monotonicity (CMON) technique, that allows to focus on the algorithmic component of OSP mechanisms.

A setting which is relatively better understood is the case of binary allocation problems, such as set systems. Here we are given a ground set of elements, each controlled by a selfish agent who privately knows the cost of the element, and a set of feasible solutions, i.e., subsets of elements in the ground set. The cost of each solution is the sum of the costs of its elements. The objective is to compute the feasible solution of minimum cost. Each agent can then be either selected or not. We normalize the utility of unselected agents to 0, whilst we use a quasi-linear utility for the selected agents, defined as the difference between the payment received from the mechanism and the cost of the element she contributes to the chosen solution. Li characterizes the class of OSP mechanisms for binary allocation problems, when the agents' domain is $[t_{\min}, t_{\max}]$, in terms of *personal clock auctions* (PCAs) – essentially, each agent faces either a descending or an ascending price auction. We are interested in the power of OSP mechanisms; in particular when can we design an OSP optimal mechanism for set systems?

To highlight the issues behind this question, let us consider a special set system, namely *path auctions*, as introduced in [19]. In this problem, each edge corresponds to a link that is owned by a selfish agent, the cost for using link i is

some private nonnegative value t_i which is known only to agent i, and the goal is to pick the shortest path between two given nodes s and t.

Can we compute the shortest path whilst guaranteeing OSP? For a graph consisting of *parallel links*, we know from [16] that the answer is yes via a simple descending auction to select the cheaper edge. Already for *slightly* more general graphs, the answer is unclear. Consider, for example, the graph in Figure 1(a). To make things even simpler let us restrict to a two-value domain $\{L, H\}$, i.e., edges cost either L or $H > 2L$. (Note that this means that we cannot rely on the PCA characterization, since our domains are not continuous.) In this setting, a simple OSP mechanism can be designed by querying the agents according to the *implementation tree* (i.e., a querying protocol where different actions are taken according to the answers received) in Figure 1(b). This algorithm is augmented with the following payments: H for edges in the selected path, 0 otherwise. It is not hard to see that, for every edge e, it is not possible that e is selected when she declares that her type is H and it is not selected when she says L. In particular, edge (s, t) is always selected when she says L, while the remaining edges are never selected when they declare H. Then, if e declares her true type, she receives a utility of $H - L$ if the true type is L and e is selected, and 0 otherwise; by inspection, she would receive at most the same utility when cheating. It turns out that this argument is enough to prove that the mechanism is indeed OSP. Does the same approach work, for example, on the *slightly* more general graph in Figure 1(c)? Consider an edge e that is queried before the type of the remaining edges is known (that is, the first edge to be queried in a sequential mechanism, or an arbitrary edge in a direct revelation mechanism). Suppose that the type of this edge is L. If she declares her type truthfully, then the worst that may occur is that the corresponding path is not selected (that occurs when this path costs $H + L$ and the alternative path costs $2L$), and thus e receives utility 0. If this edge, instead, cheats and declares H, then it is possible that the corresponding path is selected (if it costs $H+L$ and the alternative path costs $2H$) and e receives utility $H - L$. Thus, it is not obvious for an edge e lacking contingent reasoning skills, to understand that being truthful is dominant. For which graphs can we then design an OSP optimal mechanism? The goal of this work is to answer this kind of questions for set system problems.

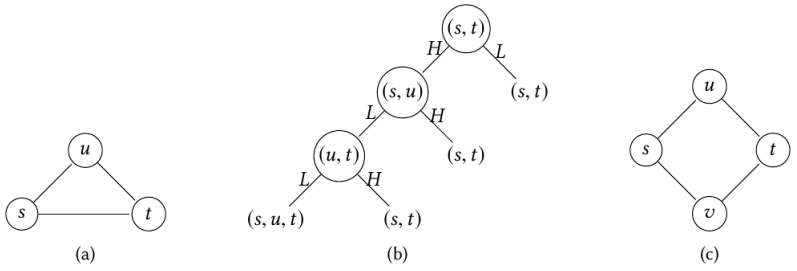

Fig. 1. Two instances of path auctions are shown in (a) and (c), while (b) is an OSP mechanism for instance (a).

Our contribution. Our main result is a complete *characterization* of OSP optimal mechanisms for set system problems (which include path auctions as a special case). To prove our results, we adopt the CMON technique developed in [9]. CMON is a powerful technique in that it allows to abstract the OSP constraints (which depend on both the solution – e.g., the shortest path – and the implementation tree) and reduces the existence of OSP mechanisms to the absence of negative-weight cycles in a carefully defined graph. This is very similar to CMON for truthful mechanisms; the difference, however, is that whilst for SP it is enough to focus on cycles of length two for essentially all domains of interest [22], for OSP, 2-cycles are in general sufficient only for small domains (of up to three-values) [9]. Hence, while the necessary condition of our characterization holds irrespectively of the domain size, since non-negative 2-cycles are always necessary, our mechanism restricts the agents to have these small domains. The technical and conceptual challenge left open is then to what extent the necessary condition are sufficient for larger domains.

Before we discuss our characterization, we exemplify our approach on a restricted version of set systems, namely path auctions on graphs comprised of two parallel paths, whose edges have two-value domains. We show how the topology (i.e., number of edges of either path) and values in the domains can change the OSP-implementability of optimal algorithms. Specifically, we show that our observation for shortest path on the graph in Figure 1(a) is not an accident as for all the graphs where a path is a direct edge, we can design an optimum OSP mechanism no matter what the alternative path looks like. Similarly, we prove that there are no OSP optimal mechanisms for the graph in Figure 1(c) and all the graphs where the two paths are composed of the same number (larger than one) of edges. As for the graphs where neither path is direct and each has a different number of edges, the existence of an OSP mechanism returning the shortest path depends on the values in the domains.

We then generalize the setting to any set system problem, wherein agents have three-value (heterogeneous) domains and fully characterize the properties needed to design OSP mechanisms.

> **Main Theorem (informal).** *There is an OSP optimal mechanism iff the set of feasible solutions are "aligned" with agent's subdomain.*

The intuition behind the characterization is simple. From OSP CMON, we know that if an OSP mechanism selects an agent e when she has a "high" cost then it must select e when she has a "low" cost (akin to monotonicity for strategyproofness). Therefore, to design an OSP optimal mechanism we need to define an implementation tree which satisfies this property. At each node of the tree, the domain of the agents is restricted to a particular subdomain, depending on the particular history; in turn, the set of possible type profiles also shrinks. Hence, there may solutions that become suboptimal for all type profiles in this set, and others that are still *alive* (i.e., optimal for at least one type profile in the set). When e is asked to separate a high cost from a low cost at node u of the tree, we then need the alive solutions to be "aligned" for the subdomain at u, which roughly means that it should never be the case that there are two bid profiles in

this subdomain for which e belongs to an optimal solution when she has a high cost and is not part of an optimal solution when she has a low cost. The somehow surprising extra aspect is that even if the alive solutions were not aligned for one single subdomain then there would be no way to design an implementation tree to bypass this misalignment.

The technical definition of alignment has some nuisance to do with the particular ways in which the OSP monotonicity can be broken, but on the positive side, rather immediately suggests how to interleave ascending and descending phases to design an OSP optimal mechanism. This characterization precisely shows how OSP needs to look at the quality of solutions among set of instances (encoded by agent subdomains) rather than just the single instance and how this is needed to inform the shape of the implementation tree. Moreover, this characterization also enables us to give a *testing algorithm*, running in time polynomial in the size of the set system instance, which flags whether an OSP optimal mechanism for the instance at hand is possible or not. This coupled with our mechanism gives a sort of automated mechanism design result in that the designer has a blackbox, comprised of testing algorithm, and possibly our mechanism, to implement the optimal solution in an OSP way.

Related work. The notion of OSP mechanism has been introduced recently by [16] and has received a lot of attention in the community. Several works have focused on understanding better the notion of OSP mechanism, and studying settings without money, namely matching and voting [2, 4, 17]. An early work on the approximation guarantee of OSP mechanisms is [10] where the authors consider OSP mechanisms for machine scheduling and facility location. A more recent study on the approximation guarantee of OSP mechanisms without money for machine scheduling is [15]. As mentioned above, a companion paper [9] introduces CMON and gives tight bounds for OSP mechanism for scheduling related machines. The use of verification [20] for OSP mechanisms is, instead, studied in [11]. The tradeoff between approximation guarantee (for machine scheduling) and relaxations of OSP is recently studied in [12].

Research in algorithmic mechanism design [13, 5] has suggested to focus on "simple" mechanisms to deal with bounded rationality. For example, posted-price mechanisms received huge attention very recently and have been applied to many different settings [1, 3, 8, 7] In these mechanisms one's own bid is immaterial for the price paid to get some goods of interest. However, posted price mechanisms do not fully capture the concept of simple mechanisms: e.g., ascending price auctions are not posted price mechanisms and still turn out to be "simple".

The automatic generation of mechanisms has been a classic desiderata in algorithmic mechanism design [23]: indeed, automated mechanisms are easier to use in practice, where inputs may quickly evolve. However, few results are known, even for SP mechanisms.

2 Preliminaries

A mechanism design setting is defined by a set of n *selfish agents* and a set of allowed *outcomes* $\mathcal{S}$. Each agent i has a *type* $t_i \in D_i$, where D_i is called the *domain* of i. The type t_i is usually assumed to be *private knowledge* of agent i. We let $t_i(X) \in \mathbb{R}$ denote the *cost* of agent i with type t_i for the outcome $X \in \mathcal{S}$.

A *mechanism* is a process for selecting an outcome $X \in \mathcal{S}$. To this aim, the mechanism interacts with agents. Specifically, agent i is observed to take *actions* (e.g., saying yes/no) that may depend on her presumed type $b_i \in D_i$ (e.g., saying yes could "signal" that the presumed type has some properties that b_i alone might enjoy). We say that agent i takes *actions compatible with (or according to)* b_i to stress this. We highlight that the presumed type b_i can be different from the real type t_i.

For a mechanism $\mathcal{M}$, we let $\mathcal{M}(\mathbf{b})$ denote the outcome returned by the mechanism when agents take actions according to their presumed types $\mathbf{b} = (b_1, \ldots, b_n)$. In our context, this outcome is given by a pair $(f, \mathbf{p})$, where $f = f(\mathbf{b})$ (termed *social choice function* or, simply, algorithm) maps the actions taken by the agents according to $\mathbf{b}$ to a feasible solution in $\mathcal{S}$, and $\mathbf{p} = \mathbf{p}(\mathbf{b}) = (p_1(\mathbf{b}), \ldots, p_n(\mathbf{b})) \in \mathbb{R}^n$ maps the actions taken by the agents according to $\mathbf{b}$ to *payments* from the mechanism to the agents.

Each selfish agent i is equipped with a *utility function* $u_i \colon D_i \times \mathcal{S} \to \mathbb{R}$. For $t_i \in D_i$ and for an outcome $X \in \mathcal{S}$ returned by a mechanism $\mathcal{M}$, $u_i(t_i, X)$ is the utility that agent i has for outcome X when her type is t_i. We define utility as a quasi-linear combination of payments and costs, i.e., $u_i(t_i, \mathcal{M}(b_i, \mathbf{b}_{-i})) = p_i(b_i, \mathbf{b}_{-i}) - t_i(f(b_i, \mathbf{b}_{-i}))$.

A mechanism $\mathcal{M}$ is *strategy-proof* (SP) if it holds that $u_i(t_i, \mathcal{M}(t_i, \mathbf{b}_{-i})) \geq u_i(t_i, \mathcal{M}(b_i, \mathbf{b}_{-i}))$ for every i, every $\mathbf{b}_{-i} = (b_1, \ldots, b_{i-1}, b_{i+1}, \ldots, b_n)$ and every $b_i \in D_i$, with t_i being the true type of i. That is, in a strategy-proof mechanism the actions taken according to the true type are dominant for each agent.

We will be focusing on *single-parameter* settings, that is, the case in which the private information of each bidder i is a single real number t_i and $t_i(X)$ can be expressed as $t_i w_i(X)$ for some publicly known function w_i.

Obvious Strategyproofness. We now formally define the concept of obviously strategy-proof (deterministic) mechanism. This concept has been introduced in [16]. However, our definition is built on the more accessible ones given in [2] and [10]. As shown in [4, 17], our definition is equivalent to Li's.[4]

Let us first formally model how a mechanism works. An *extensive-form mechanism* $\mathcal{M}$ is defined by a directed tree $\mathcal{T} = (V, E)$, called the *implementation tree*, such that:

– Every leaf ℓ of the tree is labeled with a possible outcome $X(\ell) \in \mathcal{S}$ of the mechanism;

[4] More in detail, our definition of implementation tree is equivalent to the concept of round-table mechanism in [17]. Consequently, our definition of OSP is equivalent to the concept of SP-implementation through a round table mechanism, that is proved to be equivalent to the original definition of OSP.

- Every internal vertex $u \in V$ is labeled by an agent $S(u) \in [n]$;
- Every edge $e = (u, v) \in E$ is labeled by a subset $T(e) \subseteq D = \times_i D_i$ of type profiles such that:
  - The subsets of profiles that label the edges outgoing from the same vertex u are disjoint, i.e., for every triple of vertices u, v, v' such that $(u, v) \in E$ and $(u, v') \in E$, we have that $T(u, v) \cap T(u, v') = \emptyset$;
  - The union of the subsets of profiles labelling the edges outgoing from a non-root vertex u is equal to the subset of profiles that label the edge going in u, i.e., $\bigcup_{v:\ (u,v)\in E} T(u, v) = T(\phi(u), u)$, where $\phi(u)$ is the parent of u in $\mathcal{T}$;
  - The union of the subsets of profiles that label the edges outgoing from the root vertex r is equal to the set of all profiles, i.e., $\bigcup_{v:\ (r,v)\in E} T(r, v) = D$;
  - For every u, v such that $(u, v) \in E$ and for every two profiles $\mathbf{b}, \mathbf{b}' \in T(\phi(u), u)$ such that $b_i = b'_i$, $i = S(u)$, if $\mathbf{b}$ belongs to $T(u, v)$, then $\mathbf{b}'$ must belong to $T(u, v)$ also.

Roughly speaking, the tree represents the steps of the execution of the mechanism. As long as the current visited vertex u is not a leaf, the mechanism interacts with the agent $S(u)$. Different edges outgoing from vertex u are used for modeling the different actions that agents can take during this interaction with the mechanism. As suggested above, the action that agent i takes may depend on her presumed type $b_i \in D_i$. That is, different presumed types may correspond to taking different actions, and thus to different edges. The label $T(e)$ on edge $e = (u, v)$ then lists the type profiles in which the type of $S(u)$ is one signalled by the actions assigned to e. In other words, when edge e is traversed, then the mechanism (and the other agents) can infer that the type profile must be contained in $T(e)$. The execution ends when we reach a leaf ℓ of the tree. The mechanism then returns the outcome that labels ℓ.

Observe that, according to the definition above, for every profile $\mathbf{b}$ there is only one leaf $\ell = \ell(\mathbf{b})$ such that $\mathbf{b}$ belongs to $T(\phi(\ell), \ell)$. For this reason we say that $\mathcal{M}(\mathbf{b}) = X(\ell)$. Moreover, for every type profile $\mathbf{b}$ and every node $u \in V$, we say that $\mathbf{b}$ is *compatible* with u if $\mathbf{b} \in T(\phi(u), u)$. Finally, two profiles $\mathbf{b}$, $\mathbf{b}'$ are said to *diverge* at vertex u if there are two vertices v, v' such that $(u, v) \in E$, $(u, v') \in E$ and $\mathbf{b} \in T(u, v)$, whereas $\mathbf{b}' \in T(u, v')$. For every node u in a mechanism $\mathcal{M}$ such that there are two profiles $\mathbf{b}, \mathbf{b}'$ that diverge at u, we say that u is a *divergent node*, and $i = S(u)$ the corresponding *divergent agent*. For each agent i, we define the *current domain* at node u, denoted $D_i(u)$, such that $D_i(r) = D_i$ for the root r and $D_i(u) = \cup_{\mathbf{b}\in T(\phi(u),u)} b_i$. In words, this is the set of types of i that are compatible with the actions that i took during the execution of the mechanism until node u is reached. Indeed, according to the definition above, at each node u in which i diverges, $\mathcal{M}$ partitions $D_i(u)$ in k subsets, where k is the number of children of u, and where for every child v of u, $D_i(v) \subset D_i(u)$ contains the types of bidder i compatible with the action that she takes when interacting with the mechanism at node u.

We are now ready to define obvious strategyproofness. An extensive-form mechanism $\mathcal{M}$ is *obviously strategy-proof (OSP)* if for every agent i with real type t_i, for every vertex u such that $i = S(u)$, for every $\mathbf{b}_{-i}, \mathbf{b}'_{-i}$ (with $\mathbf{b}'_{-i}$

not necessarily different from $\mathbf{b}_{-i}$), and for every $b_i \in D_i$, with $b_i \neq t_i$, such that $(t_i, \mathbf{b}_{-i})$ and $(b_i, \mathbf{b}'_{-i})$ are compatible with u, but diverge at u, it holds that $u_i(t_i, \mathcal{M}(t_i, \mathbf{b}_{-i})) \geq u_i(t_i, \mathcal{M}(b_i, \mathbf{b}'_{-i}))$. Roughly speaking, an obviously strategy-proof mechanism requires that, at each time step agent i is asked to take a decision that depends on her type, the worst utility that she can get if at this time step she behaves according to her true type is at least the best utility achievable by behaving as she had a different type. Hence, if a mechanism is obviously strategy-proof, then it is also strategy-proof. Indeed, the latter requires that truthful behavior is a dominant strategy when agents know the entire type profile, whereas the former requires that it continues to be a dominant strategy even if agents have only a partial knowledge of profiles limited to what they observed in the mechanism up to the time they are called to take their choices.

We say that an extensive-form mechanism is *trivial* if for every vertex $u \in V$ and for every two type profiles $\mathbf{b}, \mathbf{b}'$, it holds that $\mathbf{b}$ and $\mathbf{b}'$ do *not* diverge at u. That is, a mechanism is trivial if it never requires agents to take actions that depend on their type. If a mechanism is not trivial, then there is at least one divergent node. On the other hand, every execution of a mechanism (i.e., every path from the root to a leaf in the mechanism implementation tree) may go through at most $\sum_i (|D_i| - 1)$ divergent nodes, the upper bound being the case in which at each divergent node u, the agent $i = S(u)$ separates $D_i(u)$ in $D_i(u) \setminus \{b\}$ and $\{b\}$ for some $b \in D_i(u)$.

Cycle-monotonicity for OSP mechanisms. In [9], a technique – that extends the well-known cycle monotonicity for strategyproofness [21] – is introduced to study whether a mechanism is OSP. We here recall their results, needed for our characterization.

Consider an extensive-form mechanism $\mathcal{M} = (f, \mathbf{p})$ with implementation tree $\mathcal{T}$.

Definition 1 (separating vertices). *A vertex u in the implementation tree $\mathcal{T}$ is* $\alpha\beta$-separating *for agent i if the following holds: Node u is labelled with i, i.e., $i = S(u)$; there are two profiles $(\alpha, \mathbf{a}_{-i})$ and $(\beta, \mathbf{b}_{-i})$ which are compatible with u but diverge at u, where $\mathbf{a}_{-i}, \mathbf{b}_{-i} \in D_{-i}(u) = \times_{j \neq i} D_j(u)$.*

Note that there might exist several $\alpha\beta$-separating vertices for agent i as the agent may be asked to separate a from b in different paths from the root to a leaf (but only once for every such path).

Definition 2 (OSP-graph). *Let f be a social choice function and $\mathcal{T}$ be an implementation tree. We define for every agent i, the* OSP-graph $OSP_i^{(f,\mathcal{T})}$ *as follows: There is a node for each type profile in D, and a directed edge $e = ((\alpha, \mathbf{a}_{-i}), (\beta, \mathbf{b}_{-i}))$ for every $\alpha, \beta \in D_i$, $\alpha \neq \beta$, and $\mathbf{a}_{-i}, \mathbf{b}_{-i} \in D_{-i}(u)$, where u is an $\alpha\beta$-separating vertex of $\mathcal{T}$. The weight of the edge is $w(e) = \alpha(f(\beta, \mathbf{b}_{-i})) - \alpha(f(\alpha, \mathbf{a}_{-i}))$.*

Definition 3 (OSP CMON). *We say that the OSP cycle monotonicity (OSP CMON) property holds if, for all i, the graph $OSP_i^{(f,\mathcal{T})}$ does not have negative weight cycles. Moreover, we say that the OSP two-cycle monotonicity (OSP 2CMON) holds if the same is true when considering cycles of length two only, i.e., cycles with two edges only.*

We now state the relationship between OSP CMON and OSP mechanisms.

Theorem 1 ([9]). *A mechanism with implementation tree $\mathcal{T}$ is an OSP mechanism for a social function f on finite domains if and only if OSP CMON holds.*

Theorem 2 ([9]). *Let $|D_i| \leq 3$ for each agent i. A mechanism with implementation tree $\mathcal{T}$ and social choice function f is OSP iff OSP 2CMON holds.*

Set systems. In a *set system* $(E, \mathcal{F})$ we are given a set E of elements and a family $\mathcal{F} \subseteq 2^E$ of feasible subsets of E. Each element $i \in E$ is controlled by a selfish agent, that is, the cost for using i is known only to agent i and is equal to some non-negative value t_i. The social choice function f must choose a feasible subset in $\mathcal{F}$; we can use the same notation above for single-parameter agents with the restriction that $f_i(\mathbf{b}) \in \{0, 1\}$ to mean that the element controlled by agent i is either chosen by f, with $f_i(\mathbf{b}) = 1$, or not, with $f_i(\mathbf{b}) = 0$. Here our objective is social cost minimization, that is, $f^*(\mathbf{b}) \in \arg\min_{\mathbf{x}} \sum_{i=1}^{n} b_i(\mathbf{x})$. Several problems on graphs can be cast in this framework.

3 Warm-up: Shortest Path with Two-values Domains

Before stating our main results, we will illustrate our approach by providing a characterization for a simpler setting: specifically, we consider the path auction problem discussed in the introduction; this is a special case of a set system problem where the set of feasible solutions is the set of all the paths between the source node s and the destination node t in a given graph G. Moreover, we consider the case in which G has two parallel paths from the source to the destination; the first is comprised of a set T of t edges, that we will sometimes call top edges, whilst the second is comprised of a set B of b edges, that we will call bottom edges. Without loss of generality, we assume that $t \geq b$.

Proposition 1. *There is an OSP optimal mechanism for the shortest path problem on parallel paths and two-value domains $D = \{L, H\}^n$ if and only if either (1) $b = 1$ or (2) $t > b > 1$ and $\frac{H}{L} \leq \frac{t-1}{b-1}$.*

Proof (Sketch). Let us start by proving the sufficient condition. Consider the optimal mechanism that returns the bottom path in case of ties. By a case analysis, one can prove that under the hypotheses of the theorem: (i) For any top edge e, if the corresponding agent reports H, then the bottom path is chosen, i.e., $f_e(H, \mathbf{b}_{-e}) = 0$ for all $\mathbf{b}_{-e}$; (ii) For any bottom edge e, if the corresponding agent reports L, then the bottom path is chosen, i.e., $f_e(L, \mathbf{a}_{-e}) = 1$ for all $\mathbf{a}_{-e}$. Since $f_e(\cdot)$ is either 0 or 1 for this problem, the two items above imply that OSP 2CMON, and thus, by Theorem 2, OSP CMON holds for every agent e.

We next prove the necessity and show that when either (i) $t = b > 1$ or (ii) $t > b > 1$ and $\frac{H}{L} > \frac{t-1}{b-1}$, no optimal mechanism $\mathcal{M}$ can be OSP. Since $\mathcal{M}$ is optimal, it is not trivial and at some point it must separate L from H for at least one agent. We consider the first divergent agent e, and show via a simple case analysis, that OSP 2CMON is violated for this agent, thus implying that mechanism $\mathcal{M}$ is *not* OSP (Theorem 1). □

4 Set systems

In this section we characterize when OSP optimal mechanisms exist for set systems. We will formally define the concept of alignment introduced above, with a different and more technical terminology. The main message is that the feasibility of OSP optimal mechanisms depends on structural properties of the feasible solutions *and* the values in the agents' domains.

To this aim, let us first introduce the key concepts. Consider a set system problem $(E, \mathcal{F}, \mathbf{D})$. where $\mathbf{D} = (D_e)_{e \in E}$ denotes the domain. We next define some useful concepts and notation, to state our characterization and mechanism. Consider an arbitrary *subdomain* $\tilde{\mathbf{D}}$ of $\mathbf{D}$, that is, a type domain $\tilde{\mathbf{D}} = (\tilde{D}_e)_{e \in E}$ such that $\tilde{D}_e \subseteq D_e$ for all $e \in E$. We denote by $L(e, \tilde{\mathbf{D}}) = \min\{t \in \tilde{D}_e\}$ and $H(e, \tilde{\mathbf{D}}) = \max\{t \in \tilde{D}_e\}$ the lowest and the highest type for e according to the subdomain $\tilde{\mathbf{D}}$. Similarly, for any $P \subseteq E$, we let $L(P, \tilde{\mathbf{D}})$ and $H(P, \tilde{\mathbf{D}})$ be the lowest and the highest possible cost of P according to subdomain $\tilde{\mathbf{D}}$, i.e., $L(P, \tilde{\mathbf{D}}) = \sum_{e \in P} L(e, \tilde{\mathbf{D}})$ and $H(P, \tilde{\mathbf{D}}) = \sum_{e \in P} H(e, \tilde{\mathbf{D}})$. (When clear from the context, we omit the reference to $\tilde{\mathbf{D}}$ in these notations.) Finally, we let $\prec$ denote a total order among the feasible solutions in $\mathcal{F}$; this order will be used to select the optimal solution to return in case of ties.

Next concepts relate implementation trees and optimal solutions.

Definition 4 (selectable solution). *A feasible solution $P \in \mathcal{F}$ is said* selectable *for a subdomain $\tilde{\boldsymbol{D}}$ if for every other $P' \in \mathcal{F}$ it holds that $L(P \setminus P', \tilde{\boldsymbol{D}}) < H(P' \setminus P, \tilde{\boldsymbol{D}})$ or $L(P \setminus P', \tilde{\boldsymbol{D}}) = H(P' \setminus P, \tilde{\boldsymbol{D}})$ and $P \prec P'$.*

Any implementation tree gradually shrinks $\mathbf{D}$ to subdomains $\tilde{\mathbf{D}}$ by querying the agents. If the implementation tree has already shrunk to some $\tilde{\mathbf{D}}$, a selectable solution for $\tilde{\mathbf{D}}$ cannot be excluded a priori because, for some profile in $\tilde{\mathbf{D}}$, it is either the unique optimum or the optimum preferred according to the tie-breaking rule. Observe that at least one selectable solution exists for every $\tilde{\mathbf{D}}$.

While the above concept refers only to implementation tree and optimality, the next will turn out to be useful to study when there is a way to shrink $\mathbf{D}$ that returns an optimal solution but also that is compatible with OSP.

Definition 5 (strongly selectable solution). *A selectable solution P is said* strongly selectable *for a subdomain $\tilde{\boldsymbol{D}}$ if, for all $e \in P$, it continues to be selectable even for the subdomain $(\tilde{\boldsymbol{D}}_{-e}, H(e, \tilde{\boldsymbol{D}}))$, where $\tilde{\boldsymbol{D}}_{-f} = (\tilde{D}_e)_{e \neq f}$ and, with a slight abuse of notation, $H(e, \tilde{\boldsymbol{D}})$ denotes $\{H(e, \tilde{\boldsymbol{D}})\}$.*

In words, this means that solution P is still potentially optimum when any one of its elements has the largest possible cost $H(e, \tilde{\mathbf{D}})$ in $\tilde{\mathbf{D}}$.

The Analytical Characterization: Necessary Conditions. Our next two lemmas identify *necessary* conditions for the implementation tree of an OSP optimal mechanism for set systems. To this aim, we define the *obstacle domain set* $\mathcal{X}$ to contain $\mathbf{D}$ and, for each f with $|D_f| > 2$, $\tilde{\mathbf{D}}_f^{\top} = (\mathbf{D}_{-f}, \tilde{D}_f^{\top})$, where $\tilde{D}_f^{\top} = D_f \setminus L(f, \mathbf{D})$, and $\tilde{\mathbf{D}}_f^{\perp} = (\mathbf{D}_{-f}, \tilde{D}_f^{\perp})$, where $\tilde{D}_f^{\perp} = D_f \setminus H(f, \mathbf{D})$.

The first necessary condition roughly says that if there is a domain in the obstacle domain set $\mathcal{X}$ where elements of strongly selectable solutions can be excluded when they reveal their type to be as low as possible, then there is no implementation tree which yields an OSP optimal mechanism.

Lemma 1. *There is no OSP optimal mechanism for a set system problem if there is a domain $\tilde{\boldsymbol{D}} \in \mathcal{X}$ such that the following properties are both satisfied:*

(i) the set $\mathcal{S}$ of strongly selectable solutions for $\tilde{\boldsymbol{D}}$ contains at least one P with $f \in P$ such that $|\tilde{D}_f| > 1$;
(ii) for every $P \in \mathcal{S}$ and every $f \in P$ such that $|\tilde{D}_f| > 1$, there is $\bar{P}_f \in \mathcal{S}$ with $f \notin \bar{P}_f$ such that $\bar{P}_f$ remains selectable even for $(\tilde{\boldsymbol{D}}_{-f}, L(f, \tilde{\boldsymbol{D}}))$.

Proof (Sketch). Assume by contradiction that there is a domain $\tilde{\mathbf{D}} \in \mathcal{X}$ for which the conditions above are satisfied and yet there is an OSP optimal mechanism $\mathcal{M}$; let us denote with $\mathcal{T}$ its implementation tree.

Let $\mathcal{S}$ be the set of strongly selectable solutions defined in the statement, which is not empty by hypothesis. Consider the first node $u \in \mathcal{T}$ in which an agent $f \in \bigcup_{P \in \mathcal{S}} P$ diverges between $L(f, \tilde{\mathbf{D}})$ and $H(f, \tilde{\mathbf{D}})$ in the subtree compatible with the type of every agent $e \in \bigcup_{P \in \mathcal{S}} P$ being in $\tilde{D}_e$ and the type of every remaining agent e being $H(e, \tilde{\mathbf{D}})$. First, observe that, since the mechanism $\mathcal{M}$ is optimal, such a node u must exist.

Given the existence of u and f as above, we then apply the hypothesis (ii) to show a negative OSP 2-cycle. □

The second necessary property regards domains $\tilde{\mathbf{D}} \in \mathcal{X}$ for which there are solutions that are selectable but *not* strongly selectable. For each such solution P there is an agent w, that we will call the *witness* of P, such that P is no longer selectable for $\tilde{\mathbf{D}}_{hw} = (\tilde{\mathbf{D}}_{-w}, H(w, \tilde{\mathbf{D}}))$.

The next lemma intuitively says that, if there exist domains where witnesses of solutions that are selectable but not strongly selectable can be excluded (included, respectively) when they reveal their type to be the lowest (highest, respectively) possible, then there is no implementation tree which yields an OSP optimal mechanism. Its proof uses ideas similar to that of Lemma 1.

Lemma 2. *There is no OSP optimal mechanism for a set system problem if there is a domain $\tilde{\boldsymbol{D}} \in \mathcal{X}$ such that the following properties are both satisfied:*

(i) the set $\mathcal{S}$ of selectable solutions for $\tilde{\boldsymbol{D}}$ has size $|\mathcal{S}| \geq 2$, and there is at least one $P \in \mathcal{S}$ such that P is not strongly selectable;
(ii) for every f for which there is at least one selectable solution to which it belongs and at least one selectable to which it does not belong (i.e., $f \in \bigcup_{(P,P') \in \mathcal{S} \times \mathcal{S}} P \setminus P'$) both the following are true:
- *there is $\bar{P}_f \in \mathcal{S}$ s.t. $f \notin \bar{P}_f$ and $\bar{P}_f$ is selectable for $\bar{\boldsymbol{D}} = (\tilde{\boldsymbol{D}}_{-f}, L(f, \tilde{\boldsymbol{D}}))$;*
- *there is $\check{P}_f \in \mathcal{S}$ s.t. $f \in \check{P}_f$ and $\check{P}_f$ is selectable for $\check{\boldsymbol{D}} = (\tilde{\boldsymbol{D}}_{-f}, H(f, \tilde{\boldsymbol{D}}))$.*

The Analytical Characterization: The Mechanism. The two necessary conditions suggest that it is possible to design an OSP optimal mechanism when both the following properties are satisfied for *some* subdomain $\tilde{\mathbf{D}}$ containing more than one instance. When all selectable solutions are also strongly selectable then there is an f such that every P' with $f \notin P'$ ceases to be selectable if the type of f is $L(f, \tilde{\mathbf{D}})$ (this is the negation of Lemma 1). Moreover, if there is at least one selectable solution that is not strongly selectable for $\tilde{\mathbf{D}}$, then there is f such that either every P' with $f \notin P'$ ceases to be selectable if the type of f is $L(f, \tilde{\mathbf{D}})$, or every P' with $f \in P'$ ceases to be selectable if the type of f is $H(f, \tilde{\mathbf{D}})$ (this is the negation of Lemma 2). We prove that these properties are indeed sufficient by proving that Algorithm 1 admit payments for an OSP optimal mechanism, that we call $\mathcal{M}^{\mathsf{opt}}_{\mathsf{set}}$, for set systems with *three-value domains*, i.e., with $D_e \subseteq \{L_e, M_e, H_e\}$ with $L_e < M_e < H_e$ for every e.

Theorem 3. *There is an OSP optimal mechanism for a set system problem with three-value domains if and only if there is no domain $\tilde{\boldsymbol{D}} \in \mathcal{X}$ for which conditions of Lemma 1 or of Lemma 2 hold.*

The "only if" direction follows from Lemma 1 and Lemma 2. For the "if" direction, we will need the following lemma.

Lemma 3. *If the properties of Lemma 1 or of Lemma 2 are not satisfied for every domain $\tilde{\boldsymbol{D}} \in \mathcal{X}$, then they are not satisfied for every subdomain $\hat{\boldsymbol{D}}$ of $\boldsymbol{D}$.*

We are now ready to prove our main theorem.

Proof (Sketch of Theorem 3). According to Lemma 3, we can assume that for every subdomain of $\mathbf{D}$ the conditions of Lemmas 1 and 2 do not hold. The algorithm looks for an agent f we can safely ask for OSP-ness to diverge between their current $L(f)$ and $H(f)$; if f reveals type $L(f)$, then she will be securely selected, or if she reveals type $H(f)$, then she will be never selected. This shows that to each query that the mechanism does, there does not correspond a negative weight two-cycle in the OSP-graph of the queried agent. By Theorem 2 we can then conclude that $\mathcal{M}^{\mathsf{opt}}_{\mathsf{set}}$ is OSP.

Finally, the only solution in $\mathcal{P}$ at the end of the mechanism is by definition selectable for the final subdomain $\tilde{\mathbf{D}}$. To argue about optimality, we need to make sure that all the solutions excluded for bigger subdomains are not selectable for $\tilde{\mathbf{D}}$. The last key piece of the puzzle is a property of inheritance: the solutions removed for bigger subdomain, because they were not selectable, remain non-selectable for all the smaller domains. □

Note that the mechanisms $\mathcal{M}^{\mathsf{opt}}_{\mathsf{set}}$ runs in polynomial time, since it makes at most 2 queries to each agent. Interestingly, this mechanism is not a DA auction or a PCA. Indeed, it may require that single agents are involved first in an ascending phase and then in a descending phase or vice versa. In fact, it is not hard to see that this occurs even with a very simple example with only two feasible solutions, say P and Q, and three elements, x, y, and z, with $P = \{x, y\}$, and $Q = \{z\}$, with domains $D_x = D_y = D_z = \{L, M, H\}$, where $L = 1$, $M = 3$, and $H = 7$.

Input: $E, \mathcal{F}, \mathbf{D}$
Output: An optimal solution
Initialize $R = \{P \in \mathcal{F}$: P not selectable for $\mathbf{D}\}$, $\mathcal{P} = \mathcal{F} \setminus R$ and $\tilde{\mathbf{D}} = \mathbf{D}$
while $|R| < |\mathcal{F}| - 1$ **do**
  while *there is* $P \in \mathcal{P}$ *that is not strongly selectable for* $\tilde{\boldsymbol{D}}$ **do**
    if $\exists f \in \bigcup_{P \in \mathcal{P}} P$ *s.t. every* $P \in \mathcal{P}$*, with* $f \notin P$*, is not selectable for* $(\tilde{\boldsymbol{D}}_{-f}, L(f, \tilde{\boldsymbol{D}}))$ **then**
      Ask f if her type is $L(f, \tilde{\mathbf{D}})$
      if *yes* **then**
        $\tilde{\mathbf{D}} = (\tilde{\mathbf{D}}_{-f}, L(f, \tilde{\mathbf{D}}))$
        Add to R and remove from $\mathcal{P}$ every P not selectable for $\tilde{\mathbf{D}}$
      else
        $\tilde{\mathbf{D}} = (\tilde{\mathbf{D}}_{-f}, D_f \setminus L(f, \tilde{\mathbf{D}}))$
        Add to R and remove from $\mathcal{P}$ every P not selectable for $\tilde{\mathbf{D}}$
    else
      Pick $f \in \bigcup_{P \in \mathcal{P}} P$ s.t. all $P \in \mathcal{P}$, with $f \in P$, are not selectable for $(\tilde{\mathbf{D}}_{-f}, H(f, \tilde{\mathbf{D}}))$
      Ask f if her type is $H(f, \tilde{\mathbf{D}})$
      if *yes* **then**
        $\tilde{\mathbf{D}} = (\tilde{\mathbf{D}}_{-f}, H(f, \tilde{\mathbf{D}}))$
        Add to R and remove from $\mathcal{P}$ every P not selectable for $\tilde{\mathbf{D}}$
      else
        $\tilde{\mathbf{D}} = (\tilde{\mathbf{D}}_{-f}, D_f \setminus H(f, \tilde{\mathbf{D}}))$
        Add to R and remove from $\mathcal{P}$ every P not selectable for $\tilde{\mathbf{D}}$
  if $|R| < |\mathcal{F}| - 1$ **then**
    Pick $f \in \bigcup_{P \in \mathcal{P}} P$ s.t. every $P \in \mathcal{P}$, with $f \notin P$, are not selectable for $(\tilde{\mathbf{D}}_{-f}, L(f, \tilde{\mathbf{D}}))$
    Ask f if her type is $L(f, \tilde{\mathbf{D}})$
    if *yes* **then**
      $\tilde{\mathbf{D}} = (\tilde{\mathbf{D}}_{-f}, L(f, \tilde{\mathbf{D}}))$
      Add to R and remove from $\mathcal{P}$ every P that is not selectable for $\tilde{\mathbf{D}}$
    else
      $\tilde{\mathbf{D}} = (\tilde{\mathbf{D}}_{-f}, D_f \setminus L(f, \tilde{\mathbf{D}}))$
      Add to R and remove from $\mathcal{P}$ every P that is not selectable for $\tilde{\mathbf{D}}$
Return the only solution in $\mathcal{P}$

Algorithm 1: The implementation tree of the optimal algorithm for mechanism $\mathcal{M}^{\mathsf{opt}}_{\mathsf{set}}$

The Algorithmic Characterization. Note that the obstacle domain set contains at most $2|E| + 1$ domains. So we can enumerate all elements in this set in time that is polynomial in the size of the set system instance. Observe also that it takes only polynomial time (in the number of feasible solutions) to verify whether a solution is (strongly) selectable or not. Hence, the *testing algorithm*, that for every domain in the obstacle domain set checks for whether the conditions of Lemmas 1 and Lemma 2 are satisfied, is a polynomial-time algorithm.

References

1. Adamczyk, M., Borodin, A., Ferraioli, D., de Keijzer, B., Leonardi, S.: Sequential posted price mechanisms with correlated valuations. In: WINE 2015. pp. 1–15 (2015)
2. Ashlagi, I., Gonczarowski, Y.A.: Stable matching mechanisms are not obviously strategy-proof. J. Economic Theory **177**, 405–425 (2018)
3. Babaioff, M., Immorlica, N., Lucier, B., Weinberg, S.M.: A simple and approximately optimal mechanism for an additive buyer. In: FOCS 2014. pp. 21–30 (2014)
4. Bade, S., Gonczarowski, Y.A.: Gibbard-Satterthwaite success stories and obvious strategyproofness. In: EC 2017. p. 565 (2017)
5. Chawla, S., Hartline, J., Malec, D., Sivan, B.: Multi-parameter mechanism design and sequential posted pricing. In: STOC 2010. pp. 311–320 (2010)
6. Dütting, P., Gkatzelis, V., Roughgarden, T.: The performance of deferred-acceptance auctions. Math. Oper. Res. **42**(4) (2017)
7. Eden, A., Feldman, M., Friedler, O., Talgam-Cohen, I., Weinberg, S.M.: A simple and approximately optimal mechanism for a buyer with complements. In: EC 2017. pp. 323–323 (2017)
8. Feldman, M., Fiat, A., Roytman, A.: Makespan minimization via posted prices. In: EC 2017. pp. 405–422 (2017)
9. Ferraioli, D., Meier, A., Penna, P., Ventre, C.: Obviously strategyproof mechanisms for machine scheduling. In: ESA 2019 (2019)
10. Ferraioli, D., Ventre, C.: Obvious strategyproofness needs monitoring for good approximations. In: AAAI 2017. pp. 516–522 (2017)
11. Ferraioli, D., Ventre, C.: Probabilistic verification for obviously strategyproof mechanisms. In: IJCAI 2018 (2018)
12. Ferraioli, D., Ventre, C.: Obvious strategyproofness, bounded rationality and approximation: The case of machine scheduling. In: SAGT 2019 (2019)
13. Hartline, J., Roughgarden, T.: Simple versus optimal mechanisms. In: EC 2009. pp. 225–234 (2009)
14. Kagel, J., Harstad, R., Levin, D.: Information impact and allocation rules in auctions with affiliated private values: A laboratory study. Econometrica pp. 1275–1304 (1987)
15. Kyropoulou, M., Ventre, C.: Obviously strategyproof mechanisms without money for scheduling. In: AAMAS 2019 (2019)
16. Li, S.: Obviously strategy-proof mechanisms. American Economic Review **107**(11), 3257–87 (2017)
17. Mackenzie, A.: A revelation principle for obviously strategy-proof implementation. Research Memorandum 014, (GSBE) (2017)
18. Milgrom, P., Segal, I.: Deferred-acceptance auctions and radio spectrum reallocation. In: EC 2014 (2014)
19. Nisan, N., Ronen, A.: Algorithmic Mechanism Design. Games and Economic Behavior **35**, 166–196 (2001)
20. Penna, P., Ventre, C.: Optimal collusion-resistant mechanisms with verification. Games and Economic Behavior **86**, 491–509 (2014)
21. Rochet, J.C.: The taxation principle and multitime Hamilton-Jacobi equations. Journal of Mathematical Economics **14**(2), 113–128 (1985)
22. Saks, M., Yu, L.: Weak monotonicity suffices for truthfulness on convex domains. In: EC 2005. pp. 286–293 (2005)
23. Sandholm, T.: Automated mechanism design: A new application area for search algorithms. In: CP 2003. pp. 19–36 (2003)